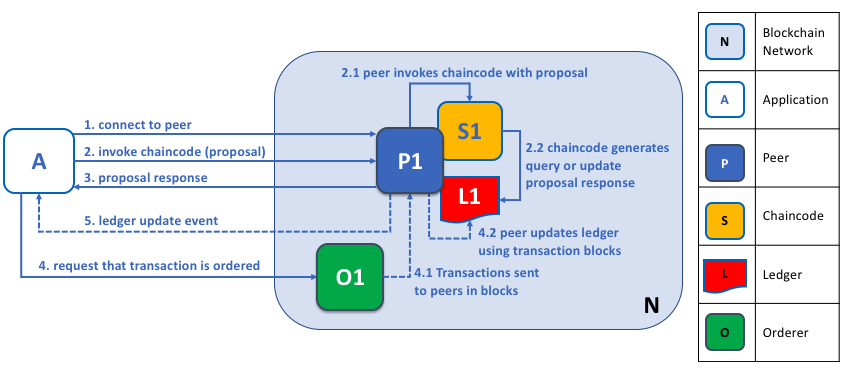
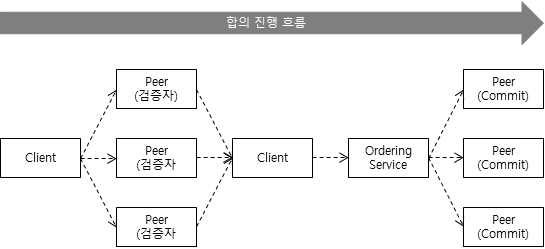
**Hyperledger Fabric**

## Node ( Peers )

* Anchor Peer
  + 다른 organization의 peer들이 서로에 대해 알기 위해 gossip protocol에서 사용된다.
  + 한 organization의 peer중 하나만 Anchor peer에 연결되어 있다면 해당 organizationd의 모든 peer는 Anchor peer와 연결되어 있는 효과를 받을 수 있다. ( gossip protocol의 통신방식 )
  + 
* Membership Service Provider (MSP, ‘org.msp’)
  + 클라이언트에게 자격 증명을 제공하고 peer가 네트워크에 참여할 수 있도록 하는 구성 요소
  + 클라이언트는 MSP에서 제공받은 자격 증명을 자신들의 트랜잭션을 증명하는데 사용한다.
  + Peer는 제공받은 자격 증명을 통해 트랜잭션 처리 결과를 인증한다 ( endorsement )
* Membership Services
  + Permissioned blockchain network에서 identities를 인증, 승인 및 관리한다.
  + Membership service는 peers와 orderers 모두에서 동작하여 blockchain operations를 인증, 승인합니다.
  + PKI-based implementation of the MSP abstraction
* Ordering Service nodes ( orderers ) -> order = 주문
  + 

# Workingflow



# Concept

* Private and Permissioned Blockchain
* Hyperledger Fabric은 네트워크 참여자를 믿을 수 있는 Membership Service Provider(MSP)를 통해 참여시킬 수 있다.
* 몇몇의 Pluggable options
  + Ledger data를 다양한 format으로 저장할 수 있다.
  + Consensus mechanisms를 변경할 수 있다.
  + Different MSPs를 지원한다.
* “Channels”를 만들 수 있다.
  + 별도의 참가자 그룹이 개별의 트랜잭션 장부를 만드는 것을 허용한다.
  + 이는 privacy transaction, data를 위해 매우 유용하다.
* Ledger
  + World state
    - Given point 시점에서의 장부의 상태를 표현한다.
    - Database of the ledger
  + Transaction log
    - 현재 world state가 되기 까지의 모든 트랜잭션을 기록한다.
    - Simply recoeds the before and after values of the ledger database
    - Update history for the world state
  + Ledger은 world state를 대체할 수 있는 데이터 저장소를 가지고 있다.
    - Default : LevelDB (Key – value ) store database
* Smart Contract
  + Chaincde로 작성된다.
  + 어플리케이션이 Ledger와 상호작용해야 할 경우 어플리케이션에 의해 호출된다.
  + 대부분의 경우, chaincode는 world state와만 querying it 과 같은 형태로 상호작용한다.( not the transaction log )
  + 현재 Go, Node.js 언어를 지원한다.

# Smart Constracts [ 기존 블록체인 플랫폼 ]

* 블록체인 플랫폼 스마트 컨트랙트 대부분은 order-execute 구조의 합의 알고리즘을 사용한다.

1. 검증되고 요청된 트랜잭션은 모든 피어 노드에 전파된다.
2. 각 피어 노드는 트랜잭션을 순차적으로 실행한다.

* order-execute 구조의 블록체인에서 스마트 컨트랙트의 실행은 반드시 완결성을 띄어야한다. 그렇지 않으면 합의에 도달할 수 없다. -> 비결정적인 실행은 제거된다 -> 이것은 광범위한 선택을 방해한다.(언어적인 선택)
* 모든 트랜잭션은 모든 노드에서 순차적으로 실행되기 때문에 Performance와 scale이 제한적이다.
* 스마트 컨트랙트 코드가 시스템 모든 노드에서 실행된다는 사실은 복잡한 측정을 요구한다. 이는 잠재적으로 발생할 수 있는 악의적인 코드로 부터 시스템 전체를 보호하고 시스템 전체를 탄력성을 보장하기 위해서 이다.

# Smart Contracts [ Hyperledger Fabric ]

* execute-order-validate architecture
  + address : resiliency, flexibility, scalability
  + 성능과 기밀 유지 문제 : 아래 3번의 단계를 통해 트랜잭션을 분리하여 order-execute 모델에 대응한다.

1. execute : 트랜잭션을 실행하고 그 정확성을 확인함으로써 해당 트랜잭션을 보증한다.
2. Order : 합의 알고리즘을 통해 트랜잭션 요청
3. Validate : 원장에 commit 되기 전에 어플리케이션별 endorsement policy에 대해 트랜잭션의 유효성을 검증한다.

* 이는 합의 전에 트랜잭션이 실행된다는 점에서 기존의 order-execute 방식과는 크게 다르다.
* 어플리케이션별 endorsement policy는 해당 스마트 컨트랙트의 옳바른 실행을 보증하기 위해 어떤 노드 혹은 얼마나 많은 노드가 보증해야하는지를 지정한다.
  + 각 트랜잭션은 해당 트랜잭션의 endorsement policy에 충족하는데 필요한 피어 노드의 하위 집합에서만 실행되면 된다. => 동시 실행을 허용하기 때문에 성능과 확장성의 증가
  + Non-determinism을 제거하였기 때문에 Fabric은 표준 프로그래밍 언어(Go, Node.js 등)를 사용할 수 있는 첫번째 블록체인 기술이다.

# Privacy and Confidentiality

* 기존 블록체인은 모든 노드에서 트랜잭션이 실행되어야 했기 때문에 컨트랙트와 해당 컨트랙트 실행을 위한 트랜잭션 데이터 또한 모든 노드가 가지고 있다. 이 때문에 네트워크의 모든 노드는 컨트랙트 코드, 트랜잭션 정보를 볼 수 있다. => 기밀성이 보장되지 않는다.
* **Public Blockchain**
  + **암호화된 데이터** : 암호화된 데이터는 기밀성을 제공하기 위한 하나의 접근법이다. 하지만 Permissionless Blockchain의 경우 모든 노드가 암호화된 데이터를 가지고 있다. 이는 암호화된 데이터를 복호화 하기에 충분한 시간과 재료를 재공하는 것이기 때문에 암호화는 유지될 수 없다.
  + **Zero Knowledge Proofs(ZKP)** : ZKP의 경우 계산하는데 상당한 시간과 계산 리소스가 필요하기 때문에 기밀성을 보장하기 위한 절충안으로 평가되고 있다.
* **Hyperledger Fabric**
  + Channel architecture을 통해 기밀성을 보장할 수 있다.
  + Fabric Network 참여자는 기본적으로 하위 그룹 사이의 채널을 형성할 수 있다. 채널은 해당 채널의 참가자들에게 트랜잭션 그룹의 visibility를 보장한다.
  + 따라서, 해당 채널에 참가자 노드만이 smart contract와 data transacted에 접근할 수 있다.
  + Fabric은 v1.2이후에 ZKP를 통한 private data 지원한다. -> private data collection을 통해

# Pluggable Consensus

* Hyperledger Fabric은 수정 가능한 consensus protocol을 지원한다.
* 현재 Fabric에서 제공하는 CFT ordering service는 Kafka와 Zookeeper를 통해 구현되었다.
* 이후에 Fabric은 Raft와 fully decentralized BFT ordering service를 통해 구현된 Raft consensus ordering service를 지원 예정이다.
* Fabric network는 다양한 ordering service를 가질 수 있지만 하나의 어플리케이션에서는 하나의 ordering service만 가능하다.

## 합의 알고리즘

* Kafka는 블록체인 합의 알고리즘이 아니다.
* Kafka는 분산 메시징 시스템으로, 실시간 대용량 로그 처리에 특화되어 있다. 따라서 다른 합의 알고리즘들이 전달되는 내용에 대한 검증을 하는 반면, kafka는 CFT(Crach Fault Tolerance) 이다. = 검증 없이 순서만 정확히 쌓이도록 한다.

## 트랜잭션과 gRPC

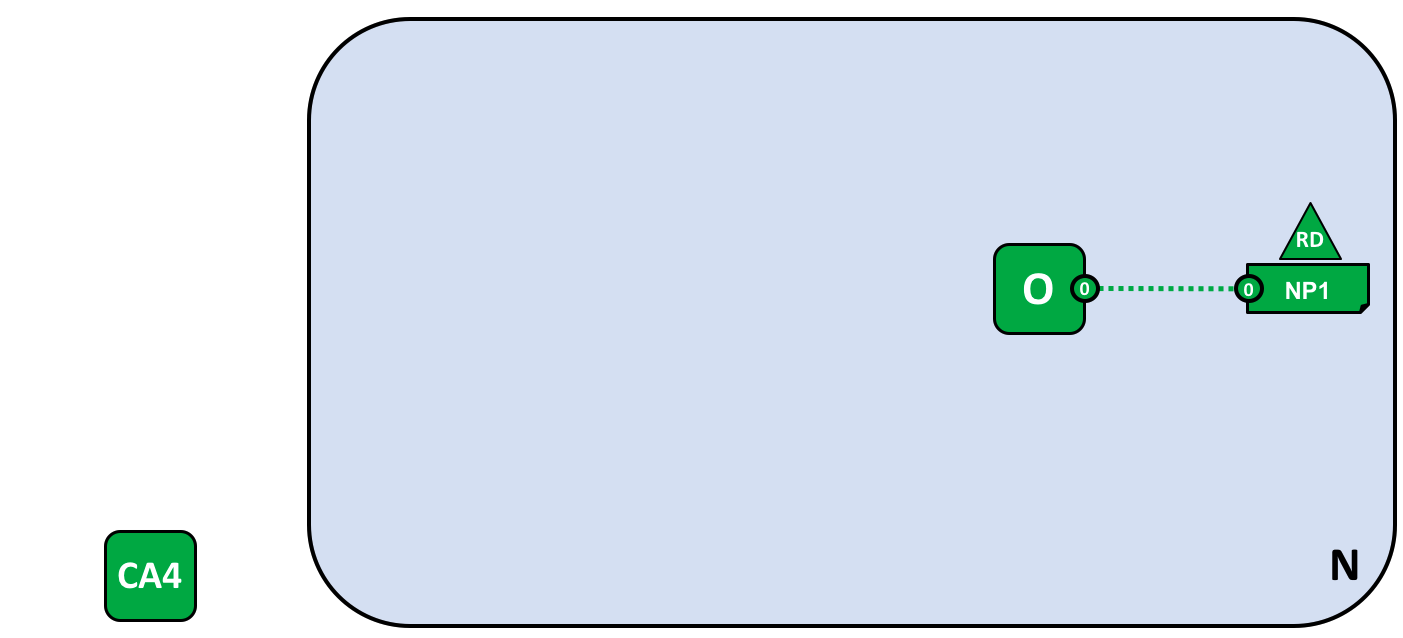
* Hyperledger Fabric은 HTTP통신이 아니라 gRPC 통신을 진행한다.
* gRPC
  + 고성능 RPC 프레임워크 [ Protocol Buffer ( 메시지 serializer )에 HTTP2 통신 방식을 입혀서 빠른 통신을 가능하도록 한다 ]
  + 어떤 메시지를, 어떤 메서드를 통해서 누가 주고 받을지를 결저한다는 것은 굉장한 특징이다. 이는 많은 노드가 많은 종류의 요청/응답을 주고 받아야할 때 유용하게 이용될 수 있다.
  + Bi-directional Streaming and Integrated Auth ( 양방향 스트리밍 통신 ) 을 지원한다.
* RPC
  + Remote Procedure Call [ 🡨🡪 HTTP 기반 REST 통신 ]
  + 요청 / 응답 측의 인터페이스 규약 ( skeleton, stub )을 정의해서 가지고 있다. 이를 기반으로 보장 정적인 통신이 가능하다.
* Protocol Buffer
  + 데이터 통신이나 저장을 위해, 데이터가 구조화되고 정렬 (serializing) 된 직렬화 데이터 구조
  + 직렬화 속도가 빠르고 직렬화된 파일의 크기가 작다.
  + JSON 파일 🡨변경🡪 프로토콜 버퍼 파일 ( 가능 )
  + 프로토콜 버퍼를 사용하기 위해선느 저장하기 위한 데이터형을 Proto file 이라는 형태로 정의한다.
  + Proto file ( \*.proto ) 🡪 protoc ( Compile ) 🡪 Source file ( \*.java, \*py, \*.go … )
* HTTP/2
  + 기본적으로 HTTP 통신은 커넥션 당 하나의 요청 밖에 처리하지 못한다. 🡪 사실상 논리적으로 하나의 로직인데 여러 번 통신이 비효율적으로 발생하는 문제가 있다. 게다가 느리다. 이런 비효율적인 HTTP 통신 방식을 개선해보고자 나온 것이 HTTP/2 이다.
  + 커넥션이 형성되면 스트리밍 형식으로 여러 데이터가 양방향으로 오고가게 된다. 지나치게 비대했던 Header도 압축되었고, 클라이언트가 요청한 데이터의 종속성을 파악하여 서버 측에서 딸린 자료들을 함께 Push할 수 있다.

# Package & Object

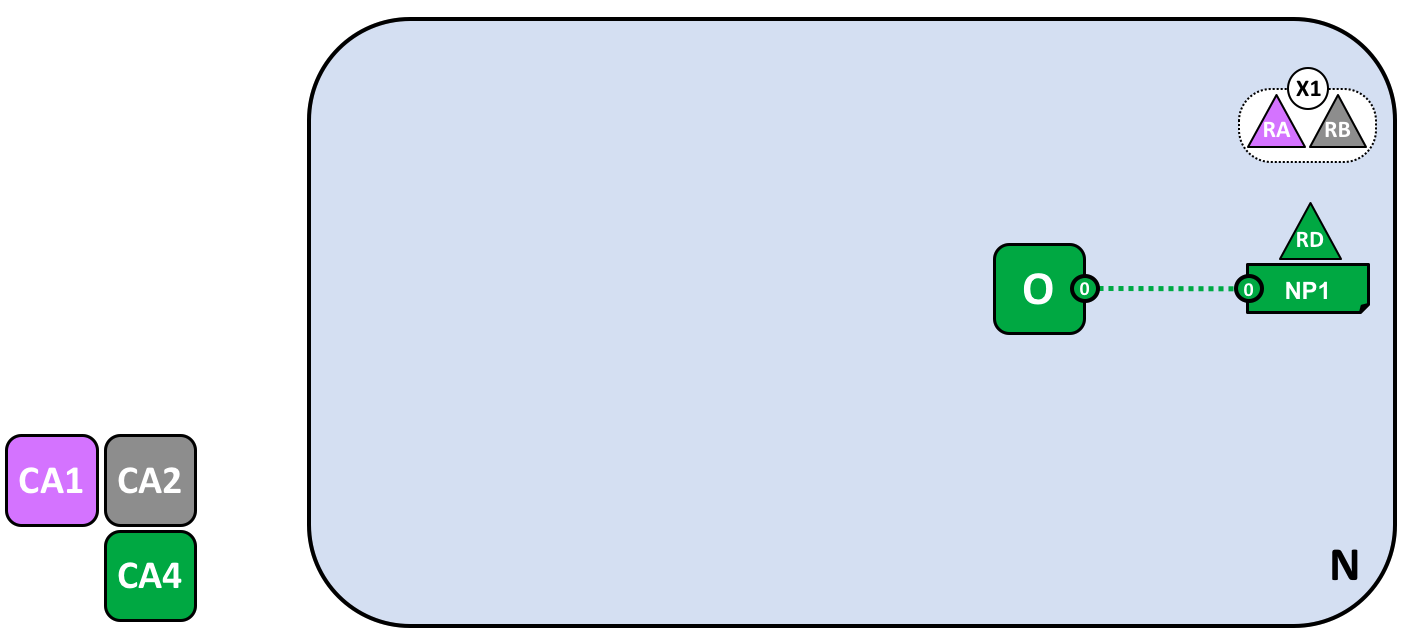
* Shim Package : 체인코드 내의 상태값, 트랜잭션 컨텍스트 등에 접근할 수 있는 API 를 제공
* Peer Package : 피어 정보를 담은 프로토콜 버퍼 패키지
* Stub Object : gRPC와 관련된 부분으로 프로토콜 버퍼 메시지 타입 호출시 이용하는 파라미터를 감싸는 객체
* Shim.ChaincodeStubInterface : SDK 내장 타입을 불러와서 가지고 있는 형태

Network With Component

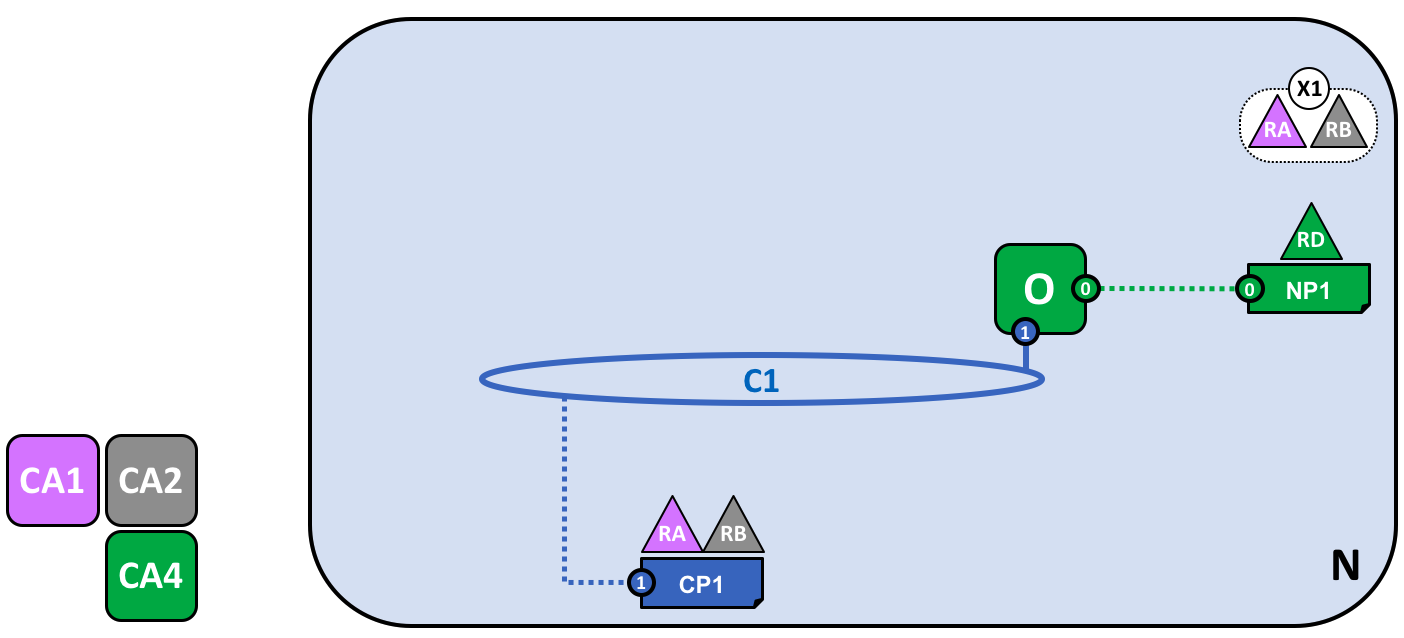
* Create Network



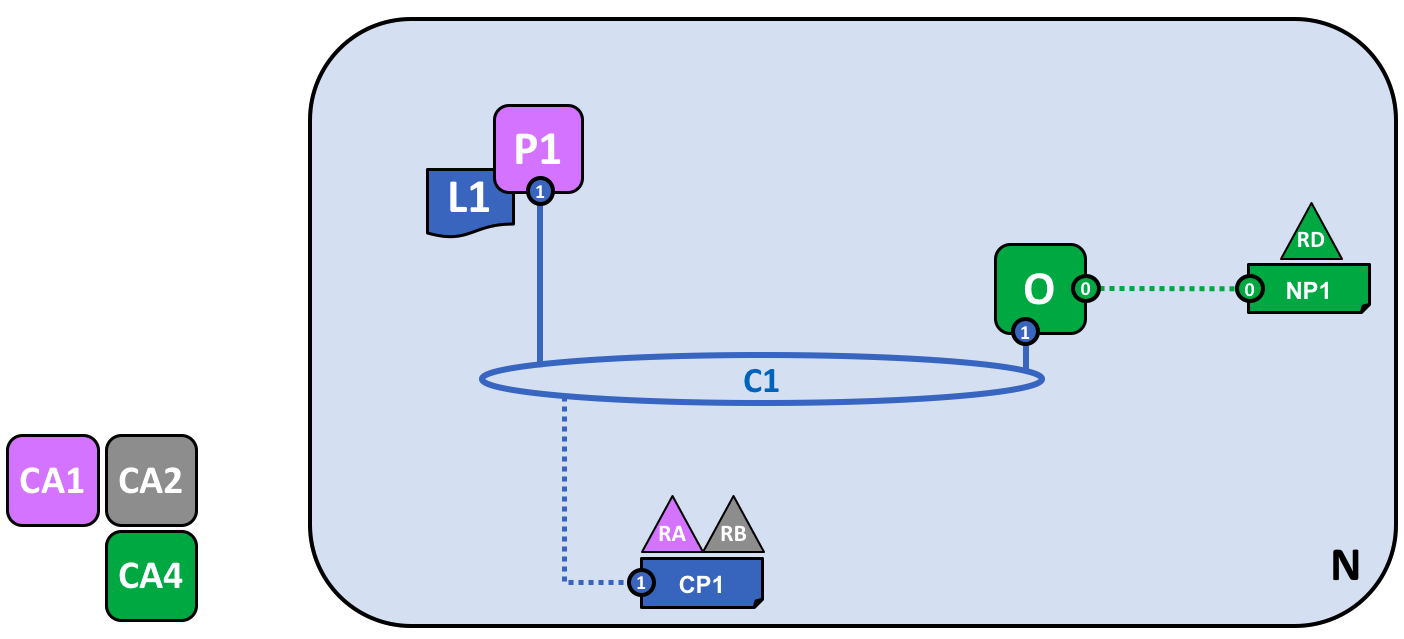
* + Ordering Service ( O ) / Network Policy ( NP1 ) / Client Application ( CA4 ) / Organization ( RD ) 으로 최초 구성
  + Ordering Service가 네트워크 내의 채널에 대한 구성 정보를 소유하고, 이를 기반으로 하여 관리자 역할을 한다. 각 채널에 대한 구성 정보는 채널에 대한 Policy 와 Membership Service를 뜻한다.
* Defining a Consortium



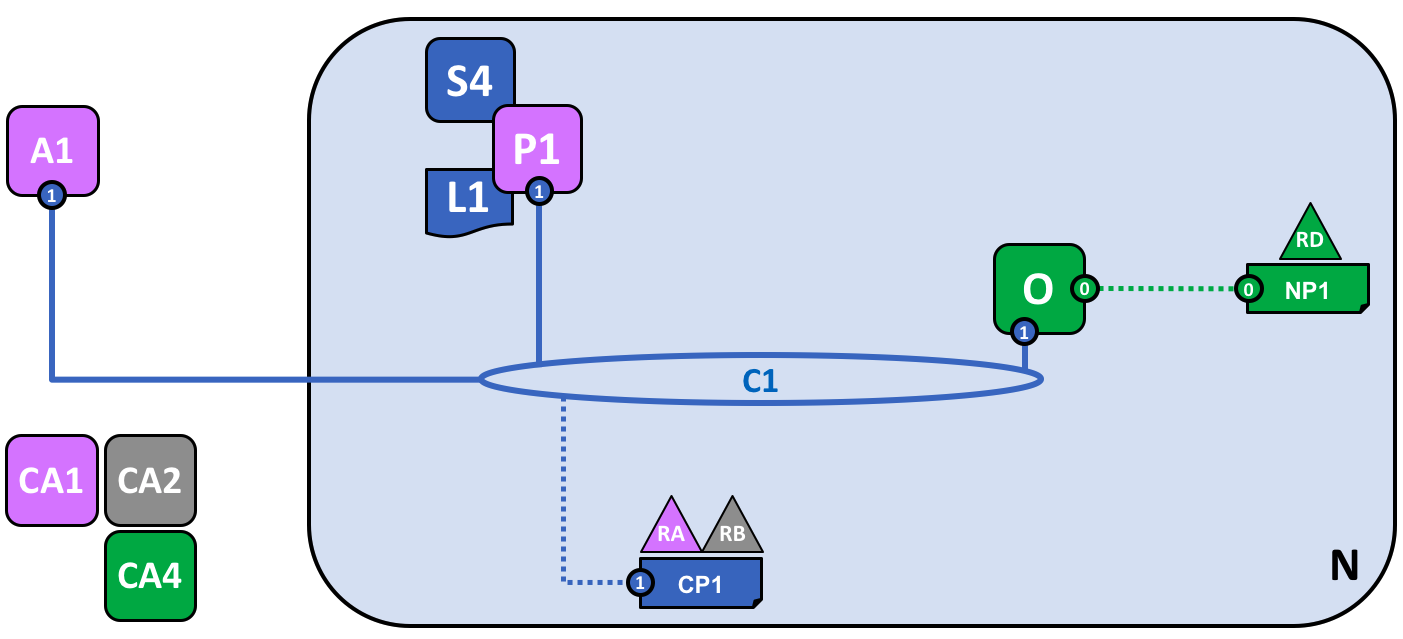
* + 네트워크 관리자는 organizations RA, RB 두 구성원이 포함된 Consortium X1 을 정의하고, 해당 컨소시엄의 정의는 network configuration NP1에 저장된다.
  + X1 컨소시엄에 포함되어 있는 구성원들은 X509 root certificate에 따라 서로 감시하고 있다.
* Create a channel for a Consortium



* + Ordering service 내에서 Config Block을 생성하는 것으로 구현되고, 이 블럭에서는 채널 자체의 구성정보를 관장한다.
  + 채널에 가입하고자 하는 조직은 이 채널의 Config에 따라 반드시 추가로 인증을 거쳐야 한다.
  + 가입 이후에도 가입한 채널의 Policy 의 관리를 받는다. 🡪 Channel Policy는 해당 채널에 참여한 Organization의 관리를 받는다.
  + 채널이 추가되면 해당 채널에 대한 CP ( Channel Policy )가 네트워크에 추가된다.
* Peer and Channel



* + Peer는 블록체인 네트워크의 실제 참여자를 의미 하며 원장을 갖고 있으며 여러 역할을 수행할 수 있다.
    - Endorsing peer : 스마트 컨트랙트에 수행될 트랜잭션을 시뮬레이션해보고 그 결과를 클라이언트 어플리케이션에 리턴해주는 피어 🡪 트랜잭션을 검증하는 역할
    - Committing Peer : Ordering까지 되어서 전달된 트랜잭션 블록을 검증하고, 검증이 완료되면 자기가 갖고 있는 원장에 해당 블록을 추가하고 관리한다. 🡪 모든 피어는 기본적으로 해당된다.
    - Anchor Peer : 채널 내의 대표 피어로써 채널에 다른 조직이 가입하면 가장 먼저 발견하는 피어
    - Leading Peer : 여러 피어를 가지고 있는 조직의 경우 ,모든 피어를 대표하여 네트워크와 커뮤니케이션하는 피어
* Application and Smart Contract



* + 스마트 컨트랙트가 피어에 반드시 설치되고 실행되어야만 클라이언트 어플리케이션이 정상 작동하고, Transaction Proposal 이 들어올 수 있다.
  + Transaction Proposal이 Client Application에서 발생하면 스마트 컨트랙트가 Endorsing Peer에 시뮬레이션을 실행하고 그 결과 (Endorsement)를 다시 어플리케이션에 반환한다. 그 결과가 Endorsement Policy에 부합하면 그 결과를 Ordering Service에 보내고 Committing Peer에 전달한다.